

Exchanging Advice and Learning to Trust

Luís Nunes¹² and Eugénio Oliveira²

¹ ISCTE, Av. Forças Armadas, 1649-026 Lisbon, Portugal, Luis.Nunes@iscte.pt

² LIACC/FEUP, Av. Dr. Roberto Frias, 4200-465, Porto, Portugal, eco@fe.up.pt

Abstract. One of the most important features of “intelligent behaviour” is the ability to learn from experience. The introduction of Multiagent Systems brings new challenges to the research in Machine Learning. New difficulties, but also new advantages, appear when learning takes place in an environment in which agents can communicate and cooperate. The main question that drives this work is “How can agents benefit from communication with their peers during the learning process to improve their individual and global performances?” We are particularly interested in environments where speed and band-width limitations do not allow highly structured communication, and where learning agents may use different algorithms. The concept of *advice-exchange*, which started out as mixture of reinforced and supervised learning procedures, is developing into a meta-learning architecture that allows learning agents to improve their learning skills by exchanging information with their peers. This paper reports the latest experiments and results in this subject.

1 Introduction

The field of Multiagent Systems (MAS) deals with software products (agents) that have some specific characteristics. The full list of these characteristics is still not consensual, but some are pointed out by most authors: the parts of the system (agents) may be distributed through several physical or logical environments; Each part has a certain degree of autonomy; Agents interact with each other. When an agent faces a problem that requires learning capabilities the above mentioned characteristics add some complexity to the learning task, but they may also provide new ways to deal with the problems. The main objective of the current work is to create an integrated set of techniques that enables learning agents to profit from these special characteristics of MAS. The most obvious way to have learning agents (hereafter referred to as agents for the sake of simplicity) profit from communication with their peers is to exchange information concerning their experience in the problem. This may sound simple, but a closer analysis brings up several questions:

- What type of information to exchange when an agent is not aware of the type of learning algorithm their peers are using?
- How does an agent select the best source of information?
- How much information can be exchanged to produce good results without over-using the (possibly limited) band-width?

- How can an agent be sure that its peer is solving a similar problem, and that the information it produces is useful?
- How should an agent decide when to explore individually and when to request information?
- How does an agent incorporate the information in its current hypothesis?

In the course of this work several tentative answers to these questions were developed. Some assumptions were made regarding the agents and the environment in which they act: it is assumed that agents will help their peers to the best of their knowledge at all times; The only feedback the environment can provide is a qualitative measure of the agent’s performance and not the desired action that it should have performed; Agents have simple communication skills and are not aware of what type of algorithm their peers use; Agents are solving problems with similar structure but possibly different dynamics, i.e. the meaning and order of the variables that compose both the environment’s state and the agent’s actions are the same in all situations, but the environment may respond differently to the same action when executed by different agents or at different times.

The most obvious way to exchange knowledge consists in sending an agent the entire set of parameters that form a solution according to a certain learning algorithm. Given the above restrictions this is not possible, both because agents are not aware of which algorithm their peers are using and because an agent’s solution may not be appropriate to another agent, if the dynamics of its environment are different. Notice that, even when in the same environment, agents may experience different dynamics due to local interaction with different peers, or localized differences in the dynamics of the environment.

Since internal solution parameters cannot be exchanged we must turn to other types of information. The simplest type of information that is already understood by all agents is the one generated by the environment, i.e. observed states, actions and quality measures. Other types of information may be used, such as “trust agent X”. The use of this information requires either that there is a previous agreement on the meaning of the messages or that an ontology for this domain is available. Because ontologies are, currently, not the main concern of this work we have adopted the first option, allowing a small number of predefined messages to be exchanged.

In the following section the reader can find an explanation of the concept of *advice-exchange*. The experimental setup is explained in section 3, followed by the presentation and discussion of the results in section 4. Section 5 is dedicated to related work and section 6 contains the concluding remarks as well a preview of the future work.

2 Advice-Exchange

Advice-Exchange, first introduced in [1], consists on giving agents the capability of requesting information to their peers and learn from it as well as from the environment’s feedback.

During the learning process agents acquire information from the environment and formulate hypothesis for the resolution of the problem they face. The key questions lie on how to select, incorporate, and benefit-from all the available information without re-processing all the data gathered by each of an agent's peers (which is time and band-width consuming). The objective is to gain knowledge based on the synthesis that was done by others without exchanging the entire solution and adopting it blindly.

In the next sections the reader can find a description of the several parts that compose the *advice-exchange* process, but before that we must define a few concepts.

An *advice* is the action that an agent (the *advisor*) would select, for a given state, based on the parameters that achieved the highest average reward in a previous epoch. The *advisee* is the agent that is actually experiencing the, above mentioned, state, and requests the advice. In some situations an *advice* comprises, both the advised action and the state which it refers to. An *epoch* is a measure of time, at the end of which a set of parameters is updated. For simplicity reasons we often relate the end of an epoch to an agent or to the environment. In this case it is possible because the learning epochs of all agents are synchronized, as well as the update of the environment's quality statistics. Nevertheless all the procedures described bellow can be equally applied in asynchronous agents. An epoch is divided in *turns*. In each turn all agents are given the opportunity to act.

When we refer to an agent's *performance* (p_n), the actual measure we are referring to is defined by:

$$p_n = \alpha r_n + (1 - \alpha)p_{n-1}, \quad (1)$$

where n is the epoch number, r_n is the *average reward* given to the agent in epoch n and $\alpha \in]0, 1[$ is a *discount parameter*. Performance measures as well as rewards are always positive values. An agent's *best performance* at epoch n is defined as:

$$b_n = \max(\beta b_{n-1}, r_n), \quad (2)$$

where $\beta \in [0.9, 1[$ is a *decay rate*. When the need arises to refer to the *absolute best performance* this will be mentioned explicitly and it will refer to the highest average reward, measured over a full epoch, that was achieved up the current moment. When it is informally mentioned that some event depends on the performance of an agent (i) being *similar to the a given agent's performance* (k), the exact definition of such a condition is:

$$p_{i,n} > \gamma p_{k,n}, \quad (3)$$

where $p_{i,n}$ is the performance of agent i in epoch n and $\gamma \in]0, 1[$ (usually close 1.0, depending on how "similar" the performance must be). A *decaying parameter* (d) is one that is updated every d_{sched} epochs (or turns, depending on the use of the parameter), according to:

$$d_n = \max(\lambda d_{n-d_{sched}}, d_{min}), \quad (4)$$

where λ is the *decay rate* and d_{min} the minimum value parameter d can reach. A *variable parameter* v is one that is updated according to:

$$v_n = \begin{cases} \min(v_{up} * v_{n-v_{sched}}, v_{max}) & : C1 \\ \max(v_{down} * v_{n-v_{sched}}, v_{min}) & : C2 \\ v_{n-v_{sched}} & : \neg C1 \wedge \neg C2. \end{cases}, \quad (5)$$

where $C1$ and $C2$ are the conditions for increase and decrease of parameter v , respectively; $v_{down} \in]0, 1[$, $v_{up} \in]1, 2[$; v_{min} and v_{max} are the minimum and maximum values parameter v is allowed to reach.

2.1 Learning Stages

When humans work in a team and learn from their peers, we can observe several stages. First they explore solutions individually until someone finds a way to solve the problem, then others observe how the problem was solved and try to mimic the solving behaviour. As this phase progresses the “students” gain a growing degree of autonomy. The “students” tend, first, to ask for a complete demonstration and, afterwards, to ask only for small pieces of information to clear the doubts concerning a particular aspect of the solution. After they have mastered the current solution the members of the group regain their autonomy and try, each individually, to improve the previous solution, or find a better one. When one succeeds the process goes back to one of the previous phases and most of the members of the group start, once again, to learn this new solution by imitation of the “teacher’s” behaviour.

This process was transposed to the domain of learning agents, by the introduction of four different learning stages:

1. Individual Exploration: Agents do not exchange information with their peers and act individually. This stage is ended at a predefined time or, optionally, when the performance of the agent stabilizes. Previous tests [1] allowed to conclude that the use of advice-exchange at an early stage will often lead all agents to explore the same area of the search-space, diminishing the exploration capabilities of the system as a whole.
2. Beginner Advisee: At this stage an agent will ask for advice for all its actions and use the same advisor for a full epoch. Agents, in this stage, have a performance that is considerably lower than the best of its peer’s.
3. Advanced Advisee: When an agent achieves a performance that is close enough to that of its advisor it becomes an Advanced Advisee and will only request advice when the choice of the next action is unclear. This is more efficient than requesting advice for all actions, providing that the agent has already reached a reasonable performance.
4. Expert: This is the stage attributed to the agent(s) with best performance(s). They may request advice in the same situation as the Advanced Advisee’s, but most of the time they are engaged in individual exploration and in trying to improve the current hypothesis by their own means.

Agents evaluate the conditions to transit from one stage to the other at the end of each epoch. An agent can transit one level down in each epoch or as many levels up as it can by meeting the appropriate conditions. A similar approach was taken in [2].

2.2 Deciding to Request Advice

When an agent is prompted to act it should choose the best action according to the quality measure it is learning to maximize. In classical Machine Learning an agent uses a learning algorithm to accumulate experience in a compact and generalizable format. This will allow it to improve its performance by choosing better actions. In MAS an agent has other options, such as requesting advice to its peers as to what is the best action to take for a given state of the environment. Advice-exchanging agents make this decision whenever they are prompted to act. The decision is based on one of two conditions: a comparison of its own performance with those of its peers, or a measure of the certainty an agent has in a given course of action. A Beginner Advisee will request advice whenever its performance is relatively low when compared to that of its peers. Other agents, that are not in the Individual Exploration stage, only request advice when they are uncertain concerning what action to take next.

When comparing its own performance with that of its peers an agent will use a *variable parameter*, labelled *self-confidence*, as a weight term to underrate or over-rate its own performance, depending if the value of this parameter is lower or higher than 1.0. The introduction of this term, as well as the use of performance measures that incorporate information from several epochs, was found useful in dealing with dynamic environments, where there may be a high variance in the reward values from one epoch to the other even when policies are stable. The *self-confidence* parameter is increased whenever the agent's performance increases or when it is similar to the best agent's performance, and decreased otherwise.

The second condition, which concerns the (un)certainly the agent has in its choice of action, is computed differently depending on the learning algorithm used and type of problem. Most learning algorithms produce an estimate of the appropriateness of each action available at a given stage. The certainty an agent has on the action to take next can be computed based on the variance of these estimates. In this case we have selected a number of the best rated options and, if the difference between each of these and the best option is lower than a given threshold, the agent is said to be uncertain regarding the next action to take. A similar technique was used in [3].

2.3 Deciding on the Source of Information

After having decided that it needs advice, the agent must choose the best source of information. In previous experiments (reported in [4,5]) it was clear that the best advisor was not always the agent with the best performance. This can happen for several reasons, but the most common of these is that advisor's and

advisee’s local environments have different dynamics and respond differently to the same actions. In partially observable environments, or when the agent is not aware of other agent’s presence, as is the case of the environments studied so far, this situation is common. In this case an agent needs to learn which of its peers it can *trust*, or who has the most useful solution for a given state.

The decision of which peer an agent (i) should request advice to, when the observed state is s , is the selection of an agent (k), where:

$$k = \operatorname{argmax}_j(\operatorname{trust}_{ij} * e_j(s)), \quad (6)$$

for all agents j , where $\operatorname{trust}_{ij}$ is a *variable parameter* that represents the level of trust an agent (i) has in peer j and $e_j(s)$ is the estimated performance agent j can achieve when the environment is in state s . The $\operatorname{trust}_{ij}$ parameter is increased for all advisors that contributed to the decisions of agent i during an epoch when the average reward is higher than the current performance, and decreased otherwise. To compute the estimated performance of an agent for a given state all agents perform an *unsupervised learning* algorithm based on the states they observe. The algorithm that was used is a simple form of *competitive learning*. This allows the agent to define clusters of states and compute the estimated performance achieved for states in a given cluster. These values are accessible to its peers upon request.

Trust in a given peer can also be influenced by another agent. When an agent issues a message of “trust agent X” the receiving agent will update the trust on agent X in the same way as if it had been an advisor for a successful epoch, thus, increasing the trust on this advisor. A similar procedure could be applied to a “distrust” message, but this was not used in the experiments.

This is an important mechanism that allows a group of agents to synchronize the requests of advice to another group in a flexible way. When it is possible to define groups of agents that are engaged in solving the same task in the same environment (i.e. are partners), it is important for unsuccessful groups, that need to learn from more successful ones, to synchronize their advice requests, so that each member of an advisee-group will request advice to a different member of the advisor-group. This way, the roles that each agent is playing in the cooperative solution can be successfully learned. When an agent issues a message to its own partners that will bias them to trust its advisor’s partners it is contributing to the synchronization of the requests. The lack of such a mechanism leads all agents to request advice to the most successful member of the advisor group, which prevents the advisee-group from learning joint strategies [5]. The difference between influencing a partner to trust others and forcing global synchronization in a team is mainly that the latter is less flexible, demands a synchronized protocol and a team-leader or a referee. The process used here allows the composition of solutions using advice from members of different successful groups when this proves efficient enough to overcome the bias towards getting advice from one group. Nevertheless, the possibility of adopting the team-synchronization is not discarded and may payoff when extra knowledge about the roles of each member of the team are known in advance.

2.4 Incorporating Advice

The process of incorporating advice depends on the learning structures used. Most learning structures (such as Artificial Neural Networks or Reinforcement Learning Quality Tables) allow some form of *supervised learning*. The details of how this is conjugated with the main learning algorithms (based on reinforcement feedback) will be explained in section 3.1.

3 Experimental Setup

This section starts with a summarized description of the learning algorithms used in the experiments. This description is focused on the variations from their standard form that were introduced in this work. For details on the standard versions of these algorithms, omitted here due to space constraints, the reader should consult the referred bibliography or [6], where a more detailed explanation of the standard algorithms and the variants used in these experiments can be found. After this first part follows a description of the experimental scenarios.

3.1 Learning Algorithms

Competitive Learning (CL) [7] is an unsupervised clustering algorithm. In this case we use simple Euclidean distance between state vectors to verify to which class a certain state belongs. During training the class representative of each observed state is moved slightly closer to that state, so that the representative will converge asymptotically to the center of mass of the set it represents. Each class representative is also associated with the estimated reward that is achieved by a given agent when acting from a state that belongs to that class. The estimated reward is updated in a similar way to the performance of an agent (equation 1). The objective is to have an estimate of how good is the performance of each advisor for a given state (or set of states).

Backpropagation (BP) [8] is a well known *supervised learning* algorithm, commonly used with networks of differentiable non-linear units connected by weights (Artificial Neural Networks, ANNs). In this work we use classical, online, *back-propagation* with a momentum term, to integrate the information received by *advice-exchange* in agents whose main learning algorithm is also based on ANNs.

Q-Learning (QL) [9] is the most commonly used learning algorithm in the class of *Reinforcement Learning*. Its most simple form (one-level Q-learning), is based on a table that stores the estimated quality $Q(s, a)$ of performing action a at state s for every possible state-action pair. When a reward r_t is received, at time t , the value of $Q(s, a)$ is updated as follows:

$$Q_{t+1}(s, a) = (1 - \alpha)Q_t(s, a) + \alpha(r_t + \beta Q_{max}(s')), \quad (7)$$

where s' is the state of the environment after performing action a at state s , α is the learning rate and $\beta \in]0, 1[$ a discount factor applied to the estimated quality of the next state ($Q_{max}(s')$), which is given by:

$$Q_{max}(s') = \max_a(Q(s', a)), \quad (8)$$

for all possible actions a when the system is in state s' .

To incorporate the advice from other agents, a form of *supervised learning* is employed in the update of the quality values. When an agent is advised to use action a as response to state s it will sum a positive value (b_{up}) to $Q(s, a)$ and a negative value (b_{down}) to all other actions available at state s (in the current experiments $b_{up} \approx na|b_{down}|$, where na is the number of actions available for the current state). A similar technique, labelled *Biasing-Binary* uses the same absolute quantity both for positive and negative feedback is reported in [10].

Evolutionary Algorithms (EAs) [11, 12] are a well known learning technique, with biological inspiration. The solution parameters are interpreted as a specimen (or phenotype), and its performance in a given problem as its fitness. After the evaluation of all the specimens the ones with best fitness are selected for breeding. The selected specimens are then mutated and crossed-over to generate a new population.

The variant of EAs used in these experiments is based on the work presented in [13], and its main characteristics are:

- The genotype is the set of real-valued matrixes that correspond to the weights an ANN of fixed size.
- Each specimen is evaluated during a certain number of epochs (3 in this case). In the first epoch of evaluation *advice-exchange* is inhibited.
- The selection strategy is elitist (keeps a number of the best specimens in the next generation).
- Contrary to [13] this variant does not use tournament selection.
- Mutation is done by disturbing all the values of the parameters with random noise with null average, and normal distribution. The variance of the mutation rate is controlled by a *decaying parameter*.
- The crossover strategy consists on choosing two parents from the selected pool and copying each node of the ANN (along with the weights of all incoming connections) from a randomly chosen parent.

Each agent contains a population of specimens. To incorporate information given by advice an agent will use backpropagation with the advised action for the current state as desired response. The selection process will work on the specimens after they have been changed by the *backpropagation* algorithm, which can be interpreted as *Lamarckian learning*. It was observed that this process took a considerable amount time to produce noticeable effects. To overcome this problem it was necessary to store the advice and replay it before the beginning of a training epoch. The advice is stored after being used. When the storage-space is filled, incoming advice will replace the oldest advice stored. The advisors are evaluated by their *trust-quality product*, defined by: $trust_{ij} * p_{i,n}$. At the beginning of an epoch, if the last advisor still has a high trust-quality product and the agent still requires advice, the stored advice is replayed. If the last advisor has lost the trust of the advisee, or its quality has diminished, the stored advice is thrown away and the next time an advisor is chosen the specimen that receives advice is re-initialized before replaying the advice.

Although advice-replay allows to incorporate knowledge gathered by advice more rapidly, is still under study because it is computationally heavy and it may reduce the diversity of the specimens.

3.2 The Pursuit Problem

The experiments were conducted in the Pursuit Problem (often referred to as Predator-Prey Problem). This problem was first introduced in [14]. This version was inspired on several variations of this environment presented in [15] and [16]. The problem faced by the predator consists in learning to catch a prey in a grid world. A predator is said to have caught the prey if at the end of a given turn it occupies the same position as the prey. The grid-world (arena) is spherical (although it is usually represented in its planar form), and contains two agents (predators) and one prey. The predators in the same arena will be referred to as *partners*. The agents receive the relative position of the prey based on their own position on the grid. The spherical shape of the grid is taken into account when computing the relative position of the prey as well as when moving. Distances are measured in the planar representation of the arena, but predators and prey can see and move across borders to the opposite sides of the arena.

The accuracy of the state received by the predator depends on its visual range. The predator perceives the correct position of the prey up to a limit defined by the visual range parameter (the value of this parameter in these experiments was 3). If the prey is at a distance greater than the visual range its relative position is disturbed (by Gaussian noise with null average) proportionally to the distance between predator and prey. The further the prey is from the limit of the visual range, the higher is the random noise added to the prey's position.

Each predator has to choose between nine possible actions in each turn, i.e. it chooses either to move in one of the eight possible directions (four orthogonal, and four diagonal), or to remain in its current position. Each predator moves one step in one direction in each turn. Any two units can occupy the same cell simultaneously at any time.

The prey, moves before the predators. To decide in which direction to move, the prey detects the presence the closest predator and moves in the opposite direction. If there is more than one predator at the same distance, one of the predators is picked randomly and the prey moves away from it regardless if it is approaching the other predator(s). The prey moves only nine out of each ten turns.

At the end of each turn predators are given a reward, which as an individual and a global component. The individual component is based on the predator's distance to the prey (d). This part of the reward is equal to 1.0, if the predator has captured the prey, or $(1.0 - d/d_{max})/10.0$ otherwise. The constant d_{max} represents the maximum distance between any two positions in the grid world and depends only on the grid dimensions. The global component of the reward is calculated by multiplying the partner's reward by a given number g , with $g < 1$.

In these experiments we used $g = 0.25$. After a successful catch the predator that caught the prey is randomly relocated in the grid-world.

This version of the predator-prey problem does not involve either explicit co-operation or competition between the predators, although emergent cooperative behaviour has been detected in some experiments, substantiating the claims reported in [17]. Predators do not sense their partner or exchange any information except for the one necessary to perform *advice-exchange*.

The predator-agents use either *Q-Learning* (QL-agents) or *Evolutionary Algorithms* (EA-agents) to learn this task. In one scenario there are also *Heuristic agents* (H-agents). H-agents are pre-coded, and perform an optimal policy for solving this problem individually, i.e. always choose the movement that will take them closer to the prey.

The scenarios used in these experiments are the following:

1. *Individual*: Four arenas, each with two predators and one prey. In each arena all predators use the same learning algorithm and they do not exchange advice. Two arenas have EA-agents, the other two have QL-agents.
2. *Social Heterogeneous*: Same as previous, except that agents may request advice to any of its seven peers in the same or other arenas.
3. *Social Heuristic*: Same as previous but with an extra arena where two H-agents are performing the same task and may also be chosen as advisors.
4. *Social Homogeneous (2 scenarios)*: Same as the *Social Heterogeneous* scenario, except that all agents use the same learning algorithm (either QL or EA).

For each of the above scenarios 11 trials were made (x4, or x8, agents of each type), each with different random seeds. Each trial ran for 20000 epochs and each epoch has 150 turns. The *Individual Exploration* stage ends at epoch 5000. For each trial there is a corresponding test which runs for 1000 epochs without learning or exchange of advice. Each agent, in the beginning of the test, loads the parameters that the corresponding agent saved during training when it achieved the best absolute score. Partnerships were kept unchanged by this procedure.

Two sets of experiments were conducted, differing only in the dimension of the grid. In the first set the grid was 10x10, and in the second 20x20.

4 Results and Discussion

The results presented in tables 1 and 2 refer to the average performance achieved in the best test (labelled Best), the average performance of the best team (labelled Team) and the average of all the (average) performances in test (labelled Avg). Each measure has the correspondent standard deviation. The measures were taken for each experiment (10x10 and 20x20), scenario (Individual, Social Homogeneous, Social Heterogeneous and Social Heuristic) and learning algorithm (QL and EA).

Table 1. Test results in Experiments 10x10 for each scenario-algorithm pair

Scenario	Alg	Best 10x10	Team 10x10	Avg 10x10
Social Heu.	Heu.	0.1762(+/-0.0196)	0.3425(+/-0.0258)	0.1730(+/-0.0055)
Social Heu.	QL	0.2062(+/-0.0167)	0.4124(+/-0.0235)	0.1942(+/-0.0027)
Social Heu.	EA	0.2059(+/-0.0158)	0.4024(+/-0.0212)	0.1883(+/-0.0046)
Social Het.	QL	0.2079(+/-0.0161)	0.4024(+/-0.0224)	0.1891(+/-0.0037)
Social Het.	EA	0.2060(+/-0.0157)	0.4098(+/-0.0208)	0.1885(+/-0.0035)
Social Hom.	QL	0.2067(+/-0.0160)	0.4048(+/-0.0234)	0.1843(+/-0.0022)
Social Hom.	EA	0.1649(+/-0.0175)	0.3255(+/-0.0224)	0.1466(+/-0.0024)
Individual	QL	0.2111(+/-0.0239)	0.4025(+/-0.0226)	0.1908(+/-0.0034)
Individual	EA	0.1617(+/-0.0150)	0.3234(+/-0.0205)	0.1429(+/-0.0041)

Table 2. Test results in Experiments 20x20 for each scenario-algorithm pair

Scenario	Alg	Best 20x20	Team 20x20	Avg 20x20
Social Heu.	Heu.	0.1222(+/-0.0123)	0.2427(+/-0.0169)	0.1216(+/-0.0034)
Social Heu.	QL	0.1203(+/-0.0118)	0.2377(+/-0.0159)	0.1177(+/-0.0029)
Social Heu.	EA	0.1213(+/-0.0119)	0.2425(+/-0.0163)	0.1179(+/-0.0030)
Social Het.	QL	0.1150(+/-0.0102)	0.2298(+/-0.0140)	0.1110(+/-0.0026)
Social Het.	EA	0.1154(+/-0.0100)	0.2308(+/-0.0134)	0.1122(+/-0.0027)
Social Hom.	QL	0.1095(+/-0.0102)	0.2190(+/-0.0142)	0.0874(+/-0.0012)
Social Hom.	EA	0.1154(+/-0.0106)	0.2308(+/-0.0146)	0.1118(+/-0.0020)
Individual	QL	0.1024(+/-0.0075)	0.1960(+/-0.0110)	0.0860(+/-0.0012)
Individual	EA	0.1319(+/-0.0158)	0.2294(+/-0.0152)	0.1071(+/-0.0026)

Concerning the main assumptions, it is important to notice that the algorithm contains the basic tools necessary to relax most of them, with the exception of exchanging advice in problems with different structures, which does not seem to be profitable unless there is some knowledge concerning the possibility of applying similar solutions to these different problems. In this case a mapping, to translate the structure of one problem to the other, would be necessary.

The assumption that agents will always help each other can be erased, possibly at the expense of slower convergence, because the agents can learn to distrust the peers that do not reply or give malicious advice, and a time-out may be introduced to deal with the lack of response of uncooperative peers. Another implicit assumption: that the epochs of all agents are synchronized; is simply a practical solution to make the epoch evaluation equivalent for all agents (they all have the same number of moves in each epoch). The underlying problem, of having untrustworthy, or noisy, evaluations from the environment, is one of the main concerns that caused the introduction of different ways to evaluate an agent's performance, thus it is being addressed.

Before we start an analysis of the results in tables 1 and 2, it is important to notice that QL and EA-agents have quite different behaviours in these two problems. In the 10x10 experiment QL-agents are clearly superior to both EA-agents

and H-agents. This is achieved by learning joint strategies in which predators push the prey towards each other.

In the 20x20 experiment joint strategies are harder to develop, and the best result is achieved by H-agents (the best learning agents are EA-agents). As can be noticed by the disparity between the average and best results for Individual EA-agents, the best individual result is uncommon. In fact there was only one agent (in 44 trials) that produced a result that was better than the H-agent's best. This has not occurred in other experiments where the best result of EA-agents, for this problem, was always slightly inferior to those of H-agents. The difference in performance of EA versus QL-agents, in these two experiments, was one of the main reasons for the choice of these variants of the problem as representatives of a more extensive set of experiments.

In the column labelled Best 10x10, in table 1, we can observe that the best result in the 10x10 experiment was achieved by Individual QL-agents (although the standard deviation is high) and the worst by Individual EA-agents. All agents trained in Social scenarios have managed to achieve a best score above 0.205, except EA-agents in Homogeneous scenarios. EA-agents clearly benefit from the advice of their QL and Heuristic peers. The average results (in column Avg 10x10) are slightly different. The best average in test was achieved by QL-agents in Social Heuristic scenarios. This is particularly interesting because H-agents cannot be responsible for teaching good joint strategies to QL-agents, since their performance is much lower and they are pre-programmed to work individually. This is a case where it seems that the exchange of information during learning can improve the average performance of agents working in a group *beyond the capabilities of the best of its members*. Another conclusion that can be drawn from these values is that *advice-exchange* seems to hurt (slightly) the performance of the best agents, although, in average, it is beneficial (this holds true also in the 20x20 experiment). This is related to the exploration-exploitation trade-off. This trade-off is present in the decision to request advice to a peer that has a slightly better solution (and be biased to explore the same search area), or explore individually, which can result in discovering an interesting area of the search-space that was not discovered by its peers. We believe that further development of the concept of *learning stages*, along with different types of conditions to make this decision, can lead to further improvements.

In the 20x20 experiment the best result was achieved by Individual EA-agents, although, as mentioned above, this is an unusual result. The worst result is that of Individual QL-agents. The Best results column shows that QL-agents benefit from advice, having better results in all Social environments than those achieved individually. In the last column, showing the average performance results for experiment 20x20, we can observe several interesting facts: the results of Individual agents are worse than any of the results in Social scenarios by the same type of agents. Also, and more important, the results in Social Heterogeneous environment for both QL and EA-agents are considerably better than the average results of their individual counterparts. Again, we have a situation where the exchange of information leads to better average results than each of the

members of the team could achieve individually. Learning from H-agents leads to the best average results. This is not surprising given that H-agents have a stable behaviour and a slightly higher performance than any of the other agent's, with the exception of the single EA-agent that achieved the best overall performance.

The results in Social Homogeneous environments are, with one exception, worse than those of the same type of agents in Heterogeneous scenarios. This gives credit to the initial assumption that heterogeneity is advantageous. These results also seem to indicate that, in most situations, the advisees have slightly lower scores than their advisors. In some cases the advisors can be disturbed by the process, possibly because they also take advice in some situations where individual exploration would be preferable in the long term.

5 Related Work

The work on exchange of information between QL-agents started in the early nineties. Whitehead [10] created a cooperative learning architecture labelled *Learning By Watching* (LBW), in which an agent learns by watching its peers' behaviour (this is equivalent to sharing series of state, action, quality triplets). In LBW an agent will use other agents' episodes as if they were its own. This work proves that the complexity of the search mechanisms of LBW is inferior to that of standard *Q-Learning* for an important class of state-spaces.

In [18], Lin uses an expert trainer to teach lessons to a QL-agent that is starting its own training. Lin experimented with several architectures, the most interesting for the current subject is: QCON-T (*Connectionist Q-Learning with Teaching*). This architecture records and replays both "taught" and "learned lessons", i.e. sequences of actions that lead to a success. Some of these sequences (the "taught lessons") are given by an expert trainer (human or automatic), while others are recorded by the student agent as it explores the problem (the "learned lessons"). The most interesting conclusion is that *"advantages of teaching should become more significant as the learning task gets more difficult"*, (section 6.4).

Tan [15] addressed the problem of exchanging information between QL-agents during the learning process. This work reports the results of sharing several types of information among agents that are working in the pursuit problem. Experiments were conducted in which QL-agents shared policies (internal solution parameters), episodes (series of state, action, quality triplets), and sensation (observed states). Tan concludes that *"sharing learned policies or episodes among agents speeds up learning at the cost of communication"* and *"for joint tasks, agents engaging in partnership can significantly outperform independent agents"*, (in *Abstract*).

In [3], Clouse reports the results of a strategy labelled *Ask for Help* (AH), in which QL-agents learn by asking other agents of the same type for suggestions and perform the suggested actions. This was labelled *learning from a teacher*. Clouse proposes two different approaches for the problem of deciding when to ask for help: in the first approach the advisee asks for help for a given percentage of the actions it has to take; in a second approach the agent asks for help only when

it is “confused” about what action to take next. Confusion is defined as having similar quality estimates for all possible actions that the agent can perform at a given stage. Clouse concludes that the integration of *learning from a teacher*, with QL achieves better results than each of its constituents separately.

Recently, quite a few researchers have focused on the problem of exchanging/sharing information during learning from several points of view [19–25]. The most important of these is the work of Price and Boutilier [24, 26] in which novice QL-agents learn by *implicit imitation* of pre-trained expert-agents. This work has some very interesting characteristics: the student agent has no knowledge of the actions done by the expert, it can only observe its state transitions; there is no explicit communication between the expert and the student; the goals and actions of both agents can be different. The student agent uses an augmented Bellman equation to calculate the utility of each state. The extra term in the Bellman equation incorporates information regarding the state transitions made by the expert. This technique has been tested in several maze problems with very good results.

6 Conclusions and Future Work

Advice-exchange has evolved over the past year, starting from simply sharing experiences with other agents, to the draft of an architecture that enables cooperation between heterogeneous groups of agents. Along the way, problems such as: disturbances in the learning process (caused by conflicting advice), learning of joint strategies from more successful groups, spurious results that caused agents to trust the “wrong” advisor, and many others, have been dealt with by introducing new concepts, mostly inspired in the human process of group learning. The initial objective was to create a process by which agents could improve their learning skills through communication with other agents. The results presented here show that this can be achieved, even though the experiments are still limited to a single problem.

Tests in more demanding problems, such as traffic-control, are being prepared, both to verify to what extent the particularities of this problem influence the process and to measure the scalability of this approach.

The, recently introduced, concept of learning stages may also be used to control the learning parameters of the agent, enabling it to vary its learning and exploration rates according to its current needs.

Other improvements, based on combination of advice from different sources, rendering unsolicited advice and the use of other learning algorithms, are also scheduled for future work.

References

1. Nunes, L., Oliveira, E.: On learning by exchanging advice. In: Proc. of the First Symposium on Adaptive Agents and Multi-Agent Systems (AAMAS/AISB’02). (2002) 29–40

2. Dorigo, M., Colombetti, M.: The role of the trainer in reinforcement learning. In et al., S.M., ed.: Proc. of MLC-COLT '94, Workshop on Robot Learning. (1994) 37–45
3. Clouse, J.A.: On integrating apprentice learning and reinforcement learning. PhD thesis, University of Massachusetts, Department of Computer Science (1997)
4. Nunes, L., Oliveira, E.: Advice-exchange in heterogeneous groups of learning agents. Technical Report 1 12/02, FEUP/LIACC (2002)
5. Nunes, L., Oliveira, E.: Advice exchange between evolutionary algorithms and reinforcement learning agents: Experimental results in the pursuit domain. In: Proc. of the Second Symposium on Adaptive Agents and Multi-Agent Systems (AAMAS/AISB'03). (2003)
6. Nunes, L., Oliveira, E.: Advice exchange architecture. Technical Report 3 04/03, FEUP/LIACC (2003)
7. Rumelhart, D.E., Zipser, D.: Feature discovery by competitive learning. *Cognitive Science* **9** (1985)
8. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. *Parallel Distributed Processing: Exploration in the Microstructure of Cognition* **1** (1986) 318–362
9. Watkins, C.J.C.H., Dayan, P.D.: Technical note: Q-learning. *Machine Learning* **8** (1992) 279–292
10. Whitehead, S.D.: A complexity analysis of cooperative mechanisms in reinforcement learning. In: Proc. of the 9th National Conf. on AI (AAAI-91). (1991) 607–613
11. Holland, J.H.: *Adaptation in Natural and Artificial Systems*. University of Michigan Press (1975)
12. Koza, J.R.: *Genetic programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge MA (1992)
13. Glickman, M., Sycara, K.: Evolution of goal-directed behavior using limited information in a complex environment. In: Proc. of the Genetic and Evolutionary Computation Conference, (GECCO-99). (1999)
14. Benda, M., Jagannathan, V., Dodhiawalla, R.: On optimal cooperation of knowledge resources. Technical Report BCS G-2012-28, Boeing AI Center, Boeing Computer Services, Bellevue, WA (1985)
15. Tan, M.: Multi-agent reinforcement learning: Independent vs. cooperative agents. In: Proc. of the Tenth Int. Conf. on Machine Learning. (1993) 330–337
16. Haynes, T., Wainwright, R., Sen, S., Schoenfeld, D.: Strongly typed genetic programming in evolving cooperation strategies. In: Proc. of the Sixth Int. Conf. on Genetic Algorithms. (1995) 271–278
17. Sen, S., Sekaran, M., Hale, J.: Learning to coordinate without sharing information. In: Proc. of the National Conf. on AI. (1994) 426–431
18. Lin, L.J.: Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning* **8** (1992) 293–321
19. Littman, M.L.: Markov games as a framework for multi-agent reinforcement learning. In: Proc. of the Eleventh International Conference on Machine Learning. (1994) 157–163
20. Thrun, S., Mitchell, T.: Lifelong robot learning. *Robotics and Autonomous Systems* **15** (1995) 25–46
21. Maclin, R., Shavlik, J.: Creating advicetaking reinforcement learners. *Machine Learning* **22** (1996) 251–281
22. Matarić, M.J.: Using communication to reduce locality in distributed multi-agent learning. Computer Science Technical Report CS-96-190, Brandeis University (1996)

23. Claus, C., Boutilier, C.: The dynamics of reinforcement learning in cooperative multiagent systems. In: Proc. of the Fifteenth National Conference on Artificial Intelligence. (1998) 746–752 Madison, WI.
24. Price, B., Boutilier, C.: Implicit imitation in multiagent reinforcement learning. In: Proc. of the Sixteenth Int. Conf. on Machine Learning. (1999) 325–334
25. Berenji, H.R., Vengerov, D.: Advantages of cooperation between reinforcement learning agents in difficult stochastic problems. In: Proc. of the Ninth IEEE International Conference on Fuzzy Systems (FUZZ-IEEE '00). (2000)
26. Price, B., Boutilier, C.: Imitation and reinforcement learning in agents with heterogeneous actions. In: Proc. of the Seventeenth Int. Conf. on Machine Learning (ICML2000). (2000)